Prior Art Literature A
(JP-B-H05-052972)

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 03-122729
(43)Date of publication of application : 24.05.1991

(51)Int.Cl. G06F 12/00

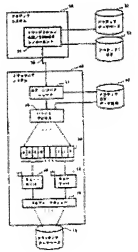
(21)Application number : 02-235953
(22)Date of filing : 07.09.1990

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>
(72)Inventor : MOHAN CHANDRASEKARAN
OBERMARCK RONALD L
TREIBER RICHARD K

(30)Priority
Priority number : 89 411729 Priority date : 25.09.1989 Priority country : US

(54) SYSTEM AND METHOD FOR UPDATING DATA BASE

(57)Abstract:
PURPOSE: To improve parallelism, to efficiently use a change content and to guarantee the consistency of a replica data base and an original data base by using peculiar ordering for recovery records on a transaction log.
CONSTITUTION: The change processing of the replica data base is realized by dividing the REDO (re-issued) records from the transaction log of the primary data base into queues. The REDO records are divided so that all the transaction records in the transfer unit (page) of the primary data base are registered in the same queues 68 and 69 in a log sequence. The queue servers 68 and 69 apply the REDO records in the queues which the servers exclusively handle to the replica data base. Thus, the replica data base becomes consistent with primary data with a lock free update mechanism for processing the pages in parallel.



⑲日本国特許庁（JP）　⑪特許出願公告

⑫特許公報（B2）　平5－52972

⑤Int. Cl.⁵	識別記号	庁内整理番号	㉔㊹公告　平成5年(1993)8月6日
G 06 F 12/00	533 J 531 D	7232－5B 7232－5B	

請求項の数 5（全11頁）

㊹発明の名称　データベース更新システム及び方法

㉑特願　平2－235953　㊺公開　平3－122729
㉒出願　平2(1990)9月7日　㊸平3(1991)5月24日
優先権主張　㉜1989年9月25日㉝米国(US)㉞411729

㉗発明者　チャンドラセカラン・モハン　アメリカ合衆国カリフォルニア州サン・ノゼ、ボーツウッド・ドライブ727番地
㉗発明者　ロナルド・エル・オバーマーク　アメリカ合衆国カリフォルニア州レッドウッド・シティ、マーリン・コート584番地
㉗発明者　リチャード・ケイ・トレーバー　アメリカ合衆国カリフォルニア州サン・ノゼ、トリーテイ・コート5018番地
㉛出願人　インターナショナル・ビジネス・マシーンズ・コーポレーション　アメリカ合衆国10504、ニューヨーク州　アーモンク（番地なし）
㉔代理人　弁理士　頓宮　孝一　外1名
審査官　赤川　誠一

㊻参考文献　特開　昭63－140352（JP，A）　特開　昭58－166459（JP，A）

1

㊼特許請求の範囲

1　レコードが転送単位で格納された第1データベースと、該第1データベースのレプリカである第2データベースと、該第1データベースに加えられた変更内容を表わすREDOレコードのシーケンスが格納されたログとを含むトランザクション処理システムにおいて、該第2データベースを、これと第1データベースとの一貫性を保つように更新するシステムであつて、

複数のキユーと、

上記第1データベースの1つの転送単位に対応するREDOレコードがすべてログ・シーケンス内の同じキユーに配置されるように、上記REDOレコードを上記キユーに配置する分配手段と、

上記各キユーのREDOレコードを上記第2データベースに適用するためにそれぞれが1つのキユーにリンクされた複数のパラレル・キユー・サーバ手段とを含む、システム。

2　請求項1に記載のデータベース更新システム

2

であつて、上記分配手段がハツシユ・プロセスであるシステム。

3　請求項1に記載のデータベース更新システムであつて、キユー・サーバ手段にリンクされたキユーの中の転送単位についてのREDOレコードによつて表される変更内容を、上記転送単位のレコードに適用することによつて、上記キユー・サーバ手段のそれぞれが上記第2データベースの転送単位を更新するシステム。

4　複数のキユー・サーバを含むトランザクション処理システムによつて実行され、第1データベースと該第1データベースのレプリカである第2データベースとの一貫性を保証する方法であつて

上記第1データベースに加えられた変更内容に対応するREDOレコードのシーケンスを累積するステツプと、

上記第1データベースの1転送単位のREDOレコードがREDOレコード・キユー1個にしか格納されないようにして、複数のREDOレコード・キ

ユーを維持するステップと、

2個以上の上記キューからのREDOレコードを上記第2データベースに並行して適用するステップとを含み、該キューのいずれか1個からのREDOレコードが、上記トランザクション処理システムの上記キュー・サーバのいずれか1個によってしか該第2データベースに適用されない、データベース更新方法。

5　ローカル・データベースを更新する方法であって、

ローカル・データベースが一貫性を保つ対象であるアクティブ・データベースからログ・レコードを受けるステップと、

どの選択された物理ストレージ・ユニットに関係するログ・レコードも1個のキューにしか格納されないような、上記ログ・レコードのキューを2個以上生成するステップと、

2個以上のサーバ・プロセスを用いることによって、2個以上の上記キューからのログ・レコードをローカル・データベースに同時適用するステップとを含み、1個のキューからのログ・レコードが、任意の時点で1個のサーバ・プロセスによってしか該データベースに適用されない、データベース更新方法。

発明の詳細な説明

A　産業上の利用分野

本発明は、トランザクション処理に関し、特に、プライマリ・データベースとレプリカ・データベースの一貫性を保つために、プライマリ・データベースのトランザクション・ログからのREDO(再実行) レコードを、レプリカ・データベースにパラレルに適用する分野に関する。

B　従来の技術

従来のトランザクション処理では、プライマリ・データベースをリモート・ロケーションで二重化できる。リモート・データベースの目的は、プライマリ・データベースの媒体に障害があつた場合に備えて、プライマリ・データベースのバックアップをとることであろう。2つのデータベースの間で一貫性を保つためには、プライマリ・データベースの処理をレプリカ・データベースに反映しなければならない。レプリカに対してレコード処理を行うときのスループットは、プライマリ・データベースを対象にしたトランザクション処理システムによつて得られるスループットと少なくとも同程度でなければならない。レプリカの処理にかかるリソースは、トランザクション処理全体にかかる場合よりも少なくする必要がある。

データベースを、プライマリ・データベースに対してトランザクション・コンシステントとし、プライマリ・データベースと同一にするためには、シリアライゼーションが必要である。プライマリ・データベースの可用性は、レプリカに対してトランザクション処理を加えることで減少することのないように維持する必要がある。

従来の技術によるデータベース二重化の具体例として、同期型分散データベースを挙げることができる。IBMのプロトタイプSystem R STARは、データベースのコピーを2箇所に格納する同期型分散データベース・システムである。ただし、現在の同期型分散データベース・システムでは、データベース相互間で一貫性を保つための処理を行うために、またデータベースのシリアライゼーションをロックとして利用するために、相当数のメッセージを交換しなければならない。

データベース媒体リカバリ処理には、よく知られたフオワード・リカバリがある。このプロセスでは、データベース・ユニットを再構成するために、ログまたはジャーナルに記録されたリカバリ・データによつて前のバージョンが更新される。その際、データベース・ユニットの元のバージョンは、ステーブル・ストレージにセーブされる。障害によつてデータベースのコピーが破壊されると、データベースの最新バージョンを再構成するために、そのユニットを最後にセーブしたバージョンがリストアされ、セーブされたバージョンに対する変更内容が、破壊されて失われたバージョンに対する変更の順序と同じ順序でログから適用される。

同期型分散データベースのフオワード・リカバリは、元のトランザクション処理を“ミラー”するものである。プライマリ・データベースの変更内容は、変更が生じたときと同じシーケンスでログされるので、ログ・データをパラレルに処理するためには、まず、最初から存在していたロックを取得し、次にログから、REDOレコードをパラレルに引き渡せばよい。この操作では、1回のリ

カバリで、所定のトランザクションに対する全レコードが処理され、そのトランザクションに対するロックは、再構成が終わるまで維持され、終了後に解除される。このリカバリ・ソリューションを制約するのは、ある特定のトランザクションに対するロックをすべて1回のプロセスで取得しなかった場合には、ロックの衝突が起こり、元のシーケンスを再現できなくなることがあるという点である。これは、ロックが取得されるまでのパラレリズムを制限し、ロックが利用できない場合は処理効率を低下させる。

米国特許出願書類第07／059666号では、ログ・レコードが、複数のキューとして、リスタート処理の間にデータベースに再適用される。この書類は、異なるデータベース・ユニットの更新は、ログに示された順序と異なる順序で適用でき、その場合でも正確さが失われることはないとしている。

Agrawalは、「マルチプロセッサ・データベース・マシンのパラレル・ロギング・アルゴリズム（A Parallel Logging Algorithm for Multi−Processor Database Machines)」、DATABASE MACHINES : Fourth International Workshop、1985年3月、の中で、複数のログに対するログ・レコードのパラレル・ライトを提案しているが、リカバリ処理の間、これらのログが1個の“バックエンド”プロセッサを通してシリアル処理される。Lyonの「二重化データベース・システムの障害防止設計(Design . Considerationsin Replicated Database Systems for Disaster Protection)、IEEE Computer Conference、1988年春、では、リモート・データベース機構が、アクテイブ・データベースのジャーナルから更新内容を抽出し、それをバックアップ用データベースに適用する。

C　課題を解決するための手段

本発明の基本を成すのは、トランザクション・ログに関するリカバリ・レコードに固有の順序づけを活用することによって、変更内容をパラレリズムを高めて効率よく適用でき、よってレプリカ・データベースと元のデータベースとの一貫性が保証されるという知見である。

D　実施例

第1図は従来のデータベース管理システムで、

データベース・レコードのトランザクション処理をサポートする。第1図のデータベース管理システムは、データベース・マネジャー（DBMGR）10を含む。マネジャー10は、ステーブル・ストレージ（DASD12など）に格納されたデータベースのアクセスと処理を管理する。管理対象のデータベースは、ステーブル・ストレージに転送単位ごとに格納される。ここで転送単位は、マネジャーがトランザクション処理のためにステーブル・ストレージから取り出す基本単位である。従来の技術では“ページ”を転送単位とするのが普通である。1ページは、物理的な境界を持つDASD12上の記憶域である。この記憶域の一つを参照符号13で示した。

データベースの転送単位は、ステーブル・ストレージ12から取り出されてアクテイブ・メモリのバッファ（BFR）14にログされる。バッファ14に入ったメモリ・ページ内のレコードが処理され、ページは、ステーブル・ストレージ12内のストレージ・セクタに返される。ステーブル・ストレージ12とバッファ14との間の転送は、バッファ・マネジャー（BMGR）16によつて処理される。マネジャー16はDBMGR10の管理下で動作する。

この基本要素はアプリケーション・プログラム18が使用できる。プログラム18はDBMGRに対してトランザクションと目的のレコードを指定する。DBMGR10は、BMGR16を呼び出して、目的のレコードを含むページをバッファ14へ転送させる。ページがバッファ14に入ると、アプリケーション18に必要なトランザクションに応じて目的のレコードが処理される。更新されたページは、ステーブル・ストレージのセクタに上書きされる。

システムに障害が起こると、バッファのページを変更するトランザクションが終了した後、更新済みページがバッファからステーブル・ストレージ12へ戻る前に、第1図のデータベース・システムの動作が停止することがある。DBMGR10のリカバリ機能が起動して修正処理が行われるのはこの場合である。

リカバリ処理では、トランザクション・ログ20をステーブル・ストレージに維持しておく必要がある。トランザクションが終了するごとに、ロ

グ２０がログ・レコードとして記録される。ログ・レコードの構成要素にはUNDO(取消)とREDOがある。UNDOは、トランザクションによって変更される前のデータベース・レコードのレプリカである。REDOは変更後のレコードのコピーである。

第２図に示すとおり、媒体リカバリ・プロセス２２は、トランザクション・ログ２０のREDOレコードを使って“フオワード”リカバリを行う。たとえば、仮にレコード信号RAないしRFで示したレコードが、ステーブル・ストレージ１２に格納されたデータ・セットに含まれる定義済みデータ・ブロックj (DDB_j) に含まれるものとする。また、DDB_jのレコードRBないしRDに対するトランザクションによって、それらのレコードが更新されるものとする。更新のため、BMGR１６はページDDB_jを実メモリのバツフア１４へ転送し、そこでレコードRBないしRDが更新されるが、いずれの場合も、レコード・データCBがCXに変更される。これらのレコードが処理されるとき、レコードが更新されたシーケンスでログ・エントリが作られる。レコードが更新されるごとに、UNDOとREDOのレコードがトランザクション・ログ２０に書き込まれる。たとえばレコードRBを更新するトランザクション・ログ・エントリには、UNDOレコードが含まれ、更新前のレコード・データを含むフイールド（CB)、レコードを識別するフイールド（RB)、およびレコードを含むステーブル・ストレージ・ページのロケーションを識別する相対ブロツク番号(RBN)がつく。ログ・レコードのREDOには、レコード・データの更新版を含むフイールド(CX)、ページを識別するフイールド（RBN)、およびレコードを識別するフイールド（RB）が含まれる。レコードRBないしRDを更新するトランザクションが終了したとすると、BMGR１６は、ページDDB_jをバツフア１４にコピーする。そこでリカバリ・プロセスは、トランザクション・ログのREDOレコードを使い、レコードRBないしRDを更新する。BMGR１６は次にバツフア・ページをDDB_jのステーブル・ストレージ・ロケーションに書き込む。

媒体リカバリ・プロセス２２は、“バツクワード”リカバリ・プロセスを伴う。このプロセスでは、終了前のトランザクションによって更新されたレコードが、UNDOレコードによって前の状態に戻される。

フオワード・リカバリとバツクワード・リカバリの両機能はIBMのデータベース製品で利用できる。この機能は、プログラム・プロダクトIBM DB2とIMS／ESAのデータベース・マネジヤーでサポートされている。これらの製品ではREDOレコードに加えてUNDOレコードもログされる。トランザクションがREDOレコードABORTSに関連する場合、UNDOレコードからのレコード・データは、REDOレコードとしてログされ、データベースに適用されるので、元の更新内容がレコードに“バツクアウト”される。IBMプログラム・プロダクトIMS／ESAのデータベース・マネジヤーFAST PATHでは、UNDOレコードではログされず、変更内容は、REDOレコードがログされて更新トランザクションが終了するまで、データベースに維持される。これに関して従来の技術では、トランザクションが無事に終了したことは、トランザクション・ログのCOMMIT(確約）レコードによって記録されるCOMMITオペレーションによって示される。レコードの変更にトランザクションが異常終了した場合、ABORT(打切り）レコードがログに入力される。

本発明を示す第３図は、アクテイブ・システムとともに動作してトラツキング・データベースを維持するトラツキング・システムである。トラツキング・データベースは、アクテイブ・システムによって維持されるプライマリ・データベースのレプリカである。アクテイブ・システム５０は(IBM　システム　370などのコンピューテイング・システムでは上述のIBMデータベース製品を構成できる)、トランザクション処理とデータベース管理を行うシステム・コンポーネント５１を持つ。コンポーネント５１は、アクテイブ・データベース５２とともにアクテイブ・ログ５３の維持、更新も行う。REDOレコードがアクテイブ・トランザクション・ログ５３に入力されると、システム５０はこれらのレコードを通信コンポーネント５５を通してトラツキング・システム６０に与える。

本発明を表すのはトラツキング・システム６０

である。システム６０は、アクテイブ・ログ５３に書き込まれたREDOレコードを、アクテイブ・ログ５３のアクテイブ・ログ・データ機構６２に入力されたシーケンスで受信するプロセス（ログ・レコード・レシート）６１を含む。機構６２はDASDなどのステーブル・ストレージで構成できる。ハッシュ・プロセス６５は、アクテイブ・ログ・データ機構６２に入力されたREDOレコードをシーケンスを追って取得し、それらを複数のREDOレコード・キュー６６に登録する。

ハッシュ・プロセス６５は従来からのものであり、この意味で、除算／剰余プロセスから構成できる。このプロセスは、各REDOレコードの相対ブロック番号に対して機能し、各レコードに対して新しいキー値を生成する。ハッシュ・プロセス６５がIMS／VSタイプのデータベース管理システムで実行させるとすると、このデータベース・システムには、REDOレコードをキュー６６から入力／出力デバイスへ移動させるAMI(アクセス・メソッド・インターフエース）が含まれる。周知のとおり、IMS AMIがシリアライズするIMS REDOレコードのサブセットは、インデツクスに新しいキー値が生成されたことを表す。新しいキー値は、ある特定のインデツクスに適用されるので、ハッシュ・プロセス６５は、所定のインデツクスに対して“新しいキー”のREDOレコードがすべてキュー６６のいずれか１個に登録されるように動作する。

キュー登録機能は、REDOレコード・キュー６６は暗示的であり、何らかの手法を用いることによつて、キュー・サーバ（後述）によるキュー登録解除処理の間に、キュー登録されたレコードを正しく順序づけることができる。ハッシュ・プロセス６５が実行されると、１個のキューの内容が、２ページ以上に隣連するREDOレコードになることがあるが、本発明では、そのキューにハッシュされるページに関連したREDOレコードがすべてキューに付加される。

また本発明では、REDOレコードがそれぞれのキューに登録され、後に、アクテイブ・トランザクション・ログ５３に置かれたのと同じシーケンスで適用される。

本発明は、これを保証するために、キュー６６のいずれか一つに対してキュー・サーバを１個し

か提供しない。すなわち、一組のレコード・キュー６６のうち、キューａ，ｂ，ｃ，ｄを扱のはキュー・サーバ６８だけである。キューｉ，ｊ，ｋを扱うのはキュー・サーバ６９だけである。したがつてレコード・キュー６６の個々のキューに対応するキュー・サーバは１個だけである。ただし１個のキュー・サーバは、２個以上のキューからのレコードを処理できる。

キュー・サーバ（サーバ６８，６９など）は、キュー６６からREDOレコードを取り出し、それらのREDOレコードを、バツフア・マネジヤー７０を通してトラツキング・データベース７２に適用する。従来のチヤネル型Ｉ／Ｏは、レコード・キュー６６からのREDOレコードを、トラツキング・データベースを含むステーブル・ストレージにバツフアする。

REDOレコードは、一度キュー６６に入ると、キュー・サーバによつてトラツキング・データベース７２に適用される。たとえばキューｂのREDOレコードは、キュー・サーバ６８によつて１個づつ取り出され、それぞれのページを更新するのに用いられる。アクテイブ・システムで生じたこれらの変更のシーケンスは、レコード転送プロセスからトラツキング・システム６０まで、アクテイブ・トランザクション・ログ５３に保存され、またアクテイブ・ログ・データセット６２にも保存される。レコードはそのレコード・ログ・シーケンスでレコード・キュー６６にハッシュされる。その結果、転送単位に関連したキュー上のREDOレコード・シーケンスによつて、各転送単位（ページ）の更新処理が時間順に正しく反映される。

第３図に示すように、キュー・サーバ６８，６９はパラレルに動作する。このパラレル動作は、たとえばマルチCPUのコンピュータで異なるCPUを用いることによつてサポートできる。キュー・サーバ６９はそれぞれ、それが扱うキューからREDOレコードを取り出して、トラツキング・データベース７２に適用する。ここでまた、キュー・サーバがIMS／VS環境で動作するとすれば、バツフア・マネジヤー７０は、サーバに呼び出されて、サーバが保持しているREDOレコードの相対ブロツク番号によつて識別されるブロツクの現在値を含むバツフアを取得する。従来のバ

ツファ・マネジャー７０は、そのプール内のブロックを発見するか、それをトラッキング・データベース７２からアクティブ・ストレージへ転送してレコードを適用するかのいずれかである。キュー・サーバは、REDOレコードをバッファ内のページに適用し、そのページをバッファ・マネジャー７０に返す。そしてページをトラッキング・データベース７２内のセクタに書き込む。

REDOレコードがトラッキング・データベース７２に適用される間、データベースは、ある時間にはトランザクションによって変更された部分だけしかページに適用されないという点で、一貫性が保たれない状態になる。このような不一致をシールドするために維持されるロックがないため、通常のデータベース・サービスは、トラッキング・データベース７２をアクセスできないようにする必要がある。障害リカバリ・アプリケーションの場合、これはデメリットとはならない。障害が起こったときだけ一貫性が保たれていればよく、トラッキング・データベース７２によるリカバリが開始されるからである。他の場合、データベースの一貫性は、アクティブ・システムに用いられるリカバリ・メカニズムを考慮した方法を採用することでいつでも確保できる。これに関して、UNDOレコードがログされないデータベース・システムでは、ハッシュ・プロセス６２がアクティブ・ログ・データベース６２からのREDOレコードを、トランザクションCOMMITが検出されるまで保持していなければならない。そこでハッシュ・プロセスは、トランザクションに関連するREDOレコードをREDOレコード・キュー６６に登録する。データベース７２をトランザクション・コンシステントな状態に保つために、アクティブ・ログ・データの処理は、REDOレコード・キューがすべて空になるまで停止される。

ログ・レコードにUNDOとREDOのレコードが含まれるデータベース・マネジャーの場合、REDOレコードは、それが届いたときにキュー登録され、したがってトラッキング・データベースに適用される。トランザクションの一貫性を保つために、アクティブ・ログ・データの処理は、REDOレコード・キューがすべて空になるまで停止され、コミットされなかった全トランザクションに対しては、標準バックアウト・ロジックを実行するために、UNDOレコードがキュー６６を通して、コミットされなかった全トランザクションに対して逆のシーケンスで適用される。

以上の点から分かるように、本発明は、トラッキング・データベースを維持するための他のプロシージャには無い大きなメリットを提供するものである。第１に、アクティブ・データベースの更新とトラッキング・データベースの更新との同期をとる必要がない。これにより可用性とパフォーマンスに関して次のようなメリットが得られる。アクティブ・データベースは、トラッキング・データベースが使用できない場合でも更新される。REDOレコードのトラッキング・データベースへの転送は、ブロックするかまたはバッチ処理でき、効率向上が図れる。また、シリアライゼーションも必要でなくなるので、コストの削減と高速化が図れる。代表的なデータベースは、入力または出力が必要なときにウエイトをサポートするため、複数のパラレル・キュー・サーバは、パラレルＩ／Ｏリソースを得て、キュー登録レコードのトラッキング・データベースへの適用を高速化できる。第３図に示すように、本発明の実施するのは容易である。当業者には明らかなように、本発明のパフォーマンスから、パス長が短くなるというメリットも得られる。ロックが必要なく、キュー・サーバのパラレリズムからスループットが向上するからである。

最適実施例

第４図は上述の本発明の最適実施例である。第４図のREDOレコード１０５は、プロセスQUEUE__REDO__RECORD(キューREDO記録）１１０に引き渡される。プロセス１１０はREDOレコード１０５をキューに引き渡し、元のトランザクション・ログ・シーケンスを保存してデータベースの正しさを保証する。プロセス１１０はハッシュ・プロシージャを伴う。これは、REDOレコード１０５のレコード・ブロック番号（RBN）を、ラベルNUMBER__OF__QUEUES（キュー数）のデータ・オブジェクト１２２の中の数ｎで割る。得られたインデックスは、WORK__QUEUE__TABLE(ワーク・キュー・テーブル）１２３とともに、特定のWORK__QUEUE__ENTRY(ワーク・キュー・エントリ）１２５を識別するのに用いられる。インデックス

13

つきのWORK__QUEUE__ENTRY１２５は、WORK__QUEUE__ENTRY１２５を先頭とする特定のレコード・キュー内の最初と最後のREDOレコード１３０へのポインタ１２７，１２９を持つ。WORK__QUEUE__ENTRY１２５はまた、キュー・サーバ・プロセス１７５のPROCESS__INFO(プロセス情報）制御ブロックを指示するSERVER__PROCESS(サーバ・プロセス）１２８を持つ。キュー・サーバ・プロセス１７５は、RESUME__STATUS(再開ステータス）フラグ１７２をサーバのPROCESS__INFO（プロセス情報）制御ブロックにセットすることによって起動される。各キュー・サーバは独自のPROCESS__INFO制御ブロックを持ち、各WORK__QUEUE__ENTRYは、そのSERVER__PROCESSフィールドによって１個のキュー・サーバしか識別しない。

表１にQUEUE__REDO__RECORDプロセス１１０を示す。

表 1

```
QUEUE__REDO__RECORD
200   QUEUE__REDO__RECORD(redo__record).
      /*入力はREDOレコード。このルーチンはログ・レコード・レシート・プロセスの下で実行*/
201   index=
      REMAINDER(redo__record.block__number、number__of__queues).
202 retry20:
203   temp=
      work__queue__table(index).work__queue__entry.last__redo.
204   redo__record.next__redo=temp.
205   CS(temp、POINTER(redo__record)、
      work__queue__table(index).work__queue__entry.last__redo).
      /*REDOレコードを、これに関連するワーク・キューからのREDOレコードのLIFO(ラストイン・フアーストアウト）チエインに自動的に追求*/
206 IF fail=ON THEN
207   GOTO retry20.
      /*アトミック・オペレーション失敗*/
```

14

```
208   IF work__queue__table(index).work__queue__entry.server__process Process__info.resume__status=OFF THEN
      RESUME(work__queue__table(index).work__queue__entry.server__process).
      /*このワーク・キューのサーバ・プロセスが保留される可能性がある場合、そのプロセスを再開して追加ワークを処理*/
*/209 END QUEUE__REDO__RECORD.
```

表１は、QUEUE__REDO__RECORDプロセス１１０を従来の疑似コードでインプリメントしたものである。このプロシージヤの機能は、REDOレコードをキュー・サーバ・プロセスに引き渡し、元のトランザクション・ログ・シーケンスを保存することにある。本発明の最適実施例では、REDOレコードがWORK__QUEUE(ワーク・キュー）に登録され、関連のキュー・サーバ・プロセスの実行が保留されているか、またはWORK__QUEUEを調べることなく保留される可能性がある場合に、その実行が再開される。

表１で、ステートメント２０１は、WORK__QUEUE__TABLE１２３のインデックスを得て特定のWORK__QUEUE__ENTRY１２５を識別する。インデックスは、REDOレコードRBNをデータ・オブジェクトNUMBER__OF__QUEUES１２２の値で割り、余りをインデックスとすることによって生成される。ハッシュ法に必要なのは、同じデータベース・ブロック（ページすなわち転送単位）に変更を加えるREDOレコードがすべて同じハッシュ値を持つことである。このハッシュ値により、これらのレコードが同じWORK__QUEUEに関連づけられる。

ステートメント２０２ないし２０７は、周知の比較・スワップ命令を用い、順序づけられたWORK__QUEUEに新しいREDOレコードを登録する。キューの順序づけはLIFOが望ましい。ステートメント２０８は、WORK__QUEUEに関連したキュー・サーバ・プロセスが、キューに追加されたばかりのワークを検出する前に、実行を保留しているかどうかをチエツクする。保留できる場合、OSのRESUME(再開）機能が発行されて、RESUME__STATUSフラグ１７２がONにセットされ、キュー・サーバ・プロセス１７５の実行が保留されているか、保留オペレーション

であれば、再開される。当業者には明らかなように、比較とスワップといったアトミック命令を用いたワークのキュー登録と再開処理を組み合わせるのは、ワークをプロセスからプロセスへ引き渡すための手法として周知のものである。プロセスの保留と再開は、従来のコンピュータのOSに提供されている標準サービスである。

ここで表2に第4図のQUEUE_SERVER(キュー・サーバ) 175を示す。

表　2

```
QUEUE_SERVER
300  QUEUE_SERVER(first_entry_index、
     number_of_entries、Process_info).
/*このルーチンはWORK_QUEUE_TABLE
のワーク・キューからのREDOレコードに適用可
能。
このルーチンが起動されるプロセスは、これら
特定のワーク・キューを扱う唯一のプロセス*/
301  last_entry_index=first_entry_index
     +number_of_entries−1.
302  top.
303  Process_info.resume_status=OFF.
304  DO index=first_entry_index TO last
     _entry_index BY 1.
305  IF work_queue_table(index). work_
     queue_entry.last_redo is not null THEN
306  DO.
     /*ワーク検出。キューにおけるこのパスの終
     わりでは保留しない*/
307  Atomically remove entire list from
     work_queue_table(index). work_queue
     _entry.last_redo.
     /*ここで取り除かれたリストはLIFOオーダ
     */
308  Reorder the list of redo records to
     FIFO order chained off of work_queue_
     table(index). work _queue _entry. first
     redo.
     /*　ここでREDOレコードは、それを処理す
     る順序でチエインされる*/
309  DO WHILE(work _ aueue _ table
     (index). work_queue_entry.first_redo is
     not equal to null).
310  Remove firstredo_record from work_
```



```
     queue_table(index). work_queue_entry.
     first_redo.
311  block _ pointer = OBTAIN _ BLOCK
     (redo _ record. database _ name、 redo _
     record.block_number).
     /*REDOレコードの変更内容が適用される物
     理プロックのコピー（メイン・メモリ内）へ
     のポインタを取得*/
312  Alter the block using redo_record.redo
     _data.
313  RELEASE_BLOCK(block_pointer).
     /*プロックをデイスク・デバイスなどのステ
     ーブル・ストレージへ書き込めるようにする
     */
314  END./*ワークを持つことが検出されたキ
     ューのREDOレコードを適用。キューをルー
     プ*/
315  END.
316  END.
317  IF process_info.resume_status=OFF
     THEN
     SUSPEND(process_info.resume_status).
     /*キュー・チエツクの間に何もキューに登録
     されなかつた場合はワークを待つ*/
318  GOTO top.
     /*このプロセスを止めるロジツクは記載しな
     い*/
319 END QUEUE_SERVER
```

REDOレコードを第3図のトラツキング・データベース72に適用するためには、キュー・サーバ・プロセスが1個以上必要になる。適当な大きさのシステムでは、アクテイブ・システム50がレコードを作成するレートに近いレートでレコードを適用するためには、複数のプロセスが必要になる。従来の技術には、システム・アクテイビテイに応じてキュー・サーバ・プロセスの個数をダイナミツクに決定・変更する機能がある。さらに、特定のワーク・キューの特定のサーバ・プロセスへの割当を変えれば、従来の機能を使って負荷を分散できる。表2は、説明の便宜上、キュー・サーバ・ロジツクを単純化している。これは、システムの初期化時に、NUMBER_OF_QUEUE_SERVER_PROCESSES(キュー・サーバ・プロセス数) 120が、キューをキュー・

サーバ・プロセスにインストールして割り当てることによって選択された値mに固定される状態が取られることを示す。

第4図は、第3図のパラレル・キュー・サーバ・アーキテクチャは示していないが、図と表2から分かるように、二重化によって第3図のトラッキング・システム60にパラレリズムを提供する1個のキュー・サーバの機能と構造を示している。

各キュー・サーバ・プロセスは、ある時点では1個以上のワーク・キューを担当するだけである。そのときキューの個数は、キュー・サーバ・プロセスの個数より多いかまたは同一である。

本発明では、単純なインクリメンタル・ループを使えば、サーバ・プロセスをスタートさせ、ワーク・キュー0からn−1（nはデータ・オブジェクト122の大きさ）の範囲の1個以上のワーク・キューに割り当てることができる。各プロセスは、関連のRROCESS_INFO制御ブロックを持ち、このブロックは、125などのWORK_QUEUE_ENTRYのフィールド128から検出できる。各プロセスは、これに制御が移ると、表2のQUEUE_SERVERを起動し、これに、ワーク・キューのインデックス値0ないしn−1（プロセスに割り当てられたもの）、MUMBER_ENTRIES(エントリ数）値1、およびそのPROCESS_INFO172を引き渡す。

先にも述べたとおり、表2は、キュー・サーバを従来の疑似コードでインプリメントしたものである。このプロシージャの機能は、REDOレコードがすべて適用されたときに、トラッキング・データベースの内容がアクティブ・データベースの内容と同じになるように、REDOレコードを、対応するトラッキング・データベース・ブロックに適用することである。

ステートメント303は、RESUME_STATUS172をOFFにして、後にプロセス110によってREDOレコードが追加された結果、レコードがキューに追加される前にそのレコードに対してプロセス175がすでにロックされていた場合には、プロセス175がRESUMEされるようにする。

ステートメント304は、プロセス175が担当するテーブル125のWORK_QUEUE_ENTRYをそれぞれ調べるため、ループを開始する。ステートメント305は、適用されるREDOレコードのWORK_QUEUE_ENTRYをチェックする。キュー登録されたREDOレコードがあれば、アトミック命令のシーケンスにより、マルチプロセス共有リスト・ヘッダLAST_REDO（最終REDO）127から全レコードが除外される。除外されれば、リストが処理されてレコードのシーケンスが逆になり、非共有リスト・ヘッダFIRST_REDO(先頭REDO）129からのレコードがチエインされる。FIRST_REDO129から始まるレコードのリスタはここで、特定のデータベース・ブロック（ページ）に対するREDOレコードを含み、変更内容は、アクテイブ・システムに適用されたのと同じシーケンスで特定ブロック（ページ）に適用される。

ステートメント309ないし314は、REDOレコードをシーケンスを追ってリストから除外し、それぞれのREDO_DATA(REDOデータ）フイールド38を、レコードのDATABASE_NAME(データベース名）フイールド134で識別されるデータベースに適用する。ステートメント311では、OBTAIN_BLOCK(ブロック獲得）機能が用いられて、所要データベース・ブロックのメイン・メモリ・コピーが検出される。OBTAIN_BLOCKとRELEASE_BLOCK(ブロック解放）の機能は、これらがバッファのプールを管理し、DASDなどの外部媒体でデータベースのステーブル・コピーをリード／ライトするという点で、代表的なデータベース・システム・バッファ管理コンポーネントである。アクテイブ・システム50のロジックは、ロックを使って、アクテイブ・データベース52内のデータベース・ブロックへのアクセスをシリアライズするが、トラッキング・システム60のロジックは、データベース・ブロックへのアクセスのシリアライゼーションを必要としない。そのデータベース・ブロックをアクセスするキュー・サーバ・プロセスは1個だけだからである。ステートメント312は、変更内容をREDO_DATAフイールド138からデータベース・ブロックのメモリ・コピーに適用する。

ステートメント317では、QUEUE_REDO_RECORDプロセス110がREDOレコードを

キューに追加登録しておらず、キュー・サーバ・プロセスをRESUME(表1のステートメント208)している場合、キュー・サーバ・プロセスがステートメント203でRESUME_STATUSをOFFにしているので、OSのSUSPEND(中断)機能が起動される。RESUMEがすでに発行されているか、または発行されたときは、ステートメント318が再び処理されて、表2のキュー・サーバが、ステートメント302までをループすることによってキューの処理を継続する。

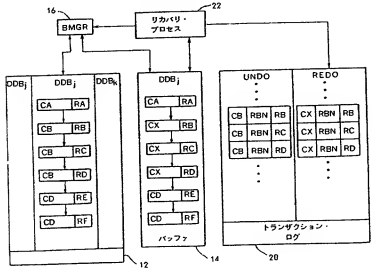
E　発明の効果

レプリカ・データベースの変更処理は、プライマリ・データベースのトランザクション・ログからのREDOレコードをキューに分けることによって達成される。REDOレコードの分割は、プライマリ・データベースの転送単位（ページ）のトランザクション・レコードがすべて、ログ・シーケンス内の同じキューに登録されるように実行される。各キュー・サーバは、これが排他的に取り扱うキュー内のREDOレコードを、レプリカ・データベースに適用する。これによりレプリカ・データベースは、そのページをパラレルに処理するロック・フリー更新メカニズムによって、プライマリ・データと一貫したものとなる。

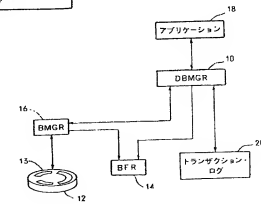
図面の簡単な説明

第1図は、データベース管理システムとしての従来のトランザクション処理システムを示す図である。第2図は、第1図のデータベース管理システムの詳細を示すブロック図である。第3図は、第1図に示す従来のデータベース管理システムと本発明との連結を示すブロック図である。第4図は、本発明で用いられるキューの構造を示す図である。

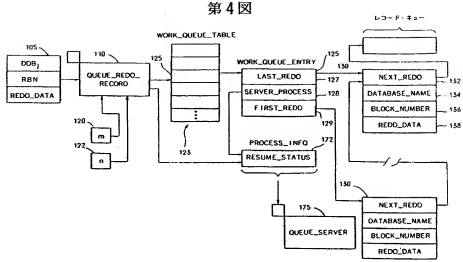
第2図



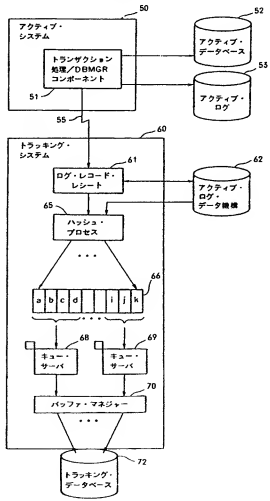
第1図



第 4 図


WORK_QUEUE_TABLE
105
DDB_j
RBN
REDO_DATA
110
QUEUE_REDO_RECORD
125
120
m
122
n
125
WORK_QUEUE_ENTRY
125
150
LAST_REDO
127
SERVER_PROCESS
128
F IRST_REDO
129
PROCESS_INFO
172
RESUME_STATUS
175
QUEUE_SERVER
レコード・キュー
NEXT_REDO
132
DATABASE_NAME
134
BLOCK_NUMBER
136
REDO_DATA
138
130
NEXT_REDO
DATABASE_NAME
BLOCK_NUMBER
REDO_DATA

第 3 図


50
アクティブ・システム
トランザクション処理/DBMGR コンポーネント
51
52
アクティブ・データベース
55
アクティブ・ログ
55
60
トラッキング・システム
61
ログ・レコード・レシート
62
アクティブ・ログ・データ機構
65
ハッシュ・プロセス
66
a b c d i j k
68
キュー・サーバ
69
キュー・サーバ
70
バッファ・マネジャー
72
トラッキング・データベース